❖ NAME

Dalia Yehia Abdelaziz Ahmed Elwakeel.

❖ PAPER ABOUT

# Simple and Efficient Pattern Matching Algorithms for Biological Sequences

❖ DR. SARA EL-METWALLY

MANSOURA UNIVERSITY
جامعة المنصورة

## ❖ ABSTRACT

The phenomenal growth of biological data is an impetus to accelerate the discovery of solutions in a wide range of computational bioinformatics domains. Pattern matching is a very useful process in various stages of the computational pipelines. for example,Pattern matching, allows users to locate specific DNA subsequences in a database or DNA chain.Any trends in these ever-expanding biological datasets are changed over time. High-speed pattern matching algorithms are needed for faster searches. The current paper presents three pattern matching algorithms that have been designed specifically to speed up searches on massive DNA sequences. The proposed algorithms improve efficiency by using word processing (rather than the character processing presented in previous works) and also by checking the sequence for the least frequent word of the pattern. In terms of time cost, the experimental results show that the presented algorithms outperform the other simulated algorithms.

## ❖ INTRODUCTION

A text, sequence, or database is scanned to detect the positions of a pattern in the text in the pattern matching problem .This type of issue must be solved for many reasons sush as image, text and signal processing and chemistry.. The pattern matching problem is prevalent in various areas of computational bioinformatics, such as simple local alignment search, biomarker discovery, sequence alignment.. In these fields, it is necessary to recognise the locations of multiple patterns in databases, including those of amino acids and nucleotides. Gene analysis and DNA sequences may be used to investigate potential illness or abnormality diagnosis in biotechnology, forensics, pathology.. DNA sequence analysis may be used to compare a gene to related genes in the same or different species also to predict its function. In a different application, the functionality of a newly discovered DNA sequence can be predicted by comparing it to existing DNA sequences. This method has been used in a number of medical trials and applications. Despite the existence of some generalised and specialised DNA pattern matching algorithms in the literature, more powerful algorithms are required. Because of their high computing costs, many existing algorithms might not be well scalable for databases or massive DNA sequences. In comparison to approximate pattern matching, the current paper focuses on the actual pattern matching problem, which identifies all occurrences of a pattern in a text. it introduces three algorithms to address the shortcomings of previous works.The algorithms was split into two phases: preprocessing and matching. During preprocessingThe text's possible intervals for matching with the pattern are identified.These candidate intervals are called windows.The windows are then carefully scanned during the matching process to be aligned with the pattern. The less windows discovered during the preprocessing phase, the less time spent in the matching phase verifying the windows.The first suggested algorithm in the current analysis seeks the windows by considering both the first and last character of the pattern at the same time. Almost all processors nowadays have a computational length of 32 or 64 bits in each execution period. To put it another way, they can handle 4 or 8 bytes of data in a split second. As a result, 4 or 8 characters denoted by a word may be compared to 4 or 8 other characters at the same time. second algorithm for performing word-based comparisons in this paper. The word processing is done using The processor's computing power. This approach produces a new class of string-matching algorithms that outperform character-based algorithms. The current work reduces the number of detected windows and speeds up the comparisons by using this approach. As a result, the efficiency in terms of time cost increases.A third algorithm that focuses on the pattern word with the fewest text repetitions( the algorithm looks for a low-frequency pattern word in the code). By reducing the number of discovered windows, this technique improves the algorithm's performance.

## RELATED WORK

Brute Force (BF) is a key approach in the pattern matching literature that does not preprocess the text or the pattern. From left to right, BF does a character-by-character analysis. After a match or a mismatch, the sliding window is moved one step to the right, and the matching is restarted from the first character of the pattern.The high consumption of time is a significant disadvantage of BF.There are approaches that merge the dynamic programming approach and DFA focused on Deterministic Finite Automata (DFA). These techniques are not always scalable for massive sequences due to the use of a finite automaton. Furthermore, because of the use of dynamic programming, the memory requirements are significantly increased. Knuth et al. introduced the KMP algorithm, which compares from the left side. In the case of a mismatch, KMP pushes the sliding window to the right by retaining the longest overlap of the paired text's suffix and the pattern's prefix. This algorithm performs linearly. Since it works well with huge alphabet sizes, the KMP algorithm has a long run time when either the alphabet size is small or the length of the pattern is short. The Boyer-Moore algorithm and its variants look for patterns in text from right to left( this algorithm begins by matching the last character of the pattern. It computes the change increment at the end of the matching character). Two useful laws (bad character and good suffix) are used to reduce the number of comparisons when a mismatch happens. The Boyer-Moore algorithm's drawback is that its preprocessing time is dependent on the pattern length and alphabet size. The Divide and Conquer Pattern Matching (DCPM) algorithm is based on comparisons. The text is checked for the pattern's rightmost character at the start of the DCPM's preprocessing step. The index of the results is saved in the character table on the right. The text is then scanned again to find the pattern's leftmost character. In similarity.The indexes are saved in the character table to the left. DCPM determines the boundary of the windows by using these two table( the elements of the tables are examined by taking into account the duration of the pattern).When the width between the windows' leftmost and rightmost characters (extracted earlier from the two tables) equals the length of the pattern , the window is located. As a result, DCPM needs two passes through the text as well as several computations to evaluate the windows. During the matching process, the algorithm examines the other windows' characters. If all of the pattern's characters and all of the text's windows align, total sameness occurs. The first algorithm in the current study promotes DCPM by recognising the windows with a single pass of the input.

## RESULTS

The performance of the proposed algorithms (FLPM, PAPM, and LFPM) is compared to that of the Brute Force (BF), Boyer-Moore (BM), and Divide and Conquer Pattern Matching (DCPM) methods. The computer environment specifications for running various simulated algorithms were as follows:

Intell®core™2 Duo CPU T6600 (a 2.2 GHz clock)

- A 2GB Memory
- Acer (Aspire 5738)
- Windows 7 ultimate 32 bit

Because the PAPM and LFPM algorithms used a 32-bit microprocessor, the word length for the PAPM and LFPM algorithms was four bytes. The HRG [29] was used as a reference in LFPM to generate the frequency table. The simulation was carried out using the C programming language. In each trial, ten patterns in the reference were searched, and the average of the findings was given. It should be noted that LFPM has a time cost while constructing the frequency table. This time overhead was addressed during the simulation because the HRG dataset was used for all experiments. There was a total of 12 milliseconds of overhead. However, because the frequency table is only constructed once for each text or database that LFPM works on, this time cost was omitted in the computations. The results of the performance evaluation of the simulated algorithms in the preprocessing, matching, and overall phases in terms of time cost are shown below.

### A. The Time of Preprocessing Phase

demonstrates the time required for the preparation step for several simulated methods across the pattern length. It should be noted that the BM time is small because the pattern is only evaluated during this algorithm's preprocessing step. This graph depicts the current study's algorithms' supremacy over the other algorithms. In FLPM, one trip through the text is enough to detect both the beginning and last character of the pattern. DCPM, on the other hand, needs two passes: one to look for the pattern's leftmost character and another to look for its rightmost. PAPM scans the text for the first word in the pattern. In a 32-bit computer, for example, the first four letters. As a result, the PAPM preprocessing step requires one pass as well. PAPM is predicted to detect fewer windows than FLPM, since it examines a word with certain letters in order to recognise the windows. FLPM, on the other hand, exclusively concentrates on two characters (the first and last). Because LFPM looks for the pattern's least frequent word, it finds fewer windows. Finding the least common word from the frequency table in LFPM, on the other hand, takes time. As a consequence, among the simulated algorithms, PAPM and LFPM spend the least amount of time. It is worth noting that BF has been excluded from this chart since it does not have a preprocessing phase.

### B. The Time of Matching Phase

The time cost of the matching step for the algorithms is presented. Because DCPM compares each element in the rightmost table to every element in the leftmost table, the matching phase takes a long time. As a result, the DCPM plot has been reduced to 3.5 milliseconds. FLPM, PAPM, and LFPM beat the other algorithms, as demonstrated. . The key reason for their supremacy is that they define a small number of windows in the preceding phase. Furthermore, by utilising word processing, PAPM and LFPM's inquiry to match the windows and the pattern is significantly faster than that of the simulated character-based algorithms, namely, BF, DCPM, BM, and FLPM.

### C. Total Time

delivers the total of the time expenses for the two stages, as seen in the preceding two figures The utilisation of word processing by PAPM and LFPM considerably decreases the time necessary to complete pattern matching, as seen in this figure. In LFPM, there is a disparity between the repetition number of the least frequent word and the repetition number of the pattern's other words. The greater the value of this difference, the greater the improvement in LFPM performance.